Université de Nice-Sophia Antipolis — Master Informatique 1ère année

Travail d'Etude et de Recherche
Communautés instantanées par Web Services



Étudiants

Michèle REYNIER
Michèle BARRÉ
Christophe ROGER
Ilya NARAGHI
Jean-Michael LEGAIT

Encadrants

Philippe COLLET
Hervé CHANG



2005–2006

Table des matières

1 Introduction **1**
1.1 Objectifs . . . 1
1.2 Principes . . . 1
1.3 Architecture générale . . . 2

2 Cahier des charges **5**
2.1 Connectivité client/serveur . . . 5
2.1.1 Création d'un couple compte/mot de passe . . . 5
2.1.2 Sauvegarde des mots clés pour le client lourd . . . 5
2.2 Création dynamique des communautés instantanées . . . 5
2.2.1 Analyse des mots clés . . . 5
2.2.2 Affectation aux salons . . . 5
2.2.3 Notification . . . 6
2.3 Administration . . . 6
2.3.1 Plugin ou module pour Wildfire . . . 6
2.3.2 Console d'administration . . . 6
2.3.3 Console de test . . . 6

3 Planning du projet **7**
3.1 Client . . . 8
3.2 Serveur . . . 9
3.3 Documentation et préparation à la soutenance . . . 10
3.3.1 Rédaction de la documentation . . . 10
3.3.2 Préparation de la soutenance . . . 10

4 Travail réalisé **11**
4.1 Architecture logicielle . . . 11
4.1.1 Architecture générale . . . 11
4.1.2 Déploiement sur Tomcat des applications Axis et Wildfire . . . 12
4.2 Client . . . 12
4.2.1 Client léger . . . 12
4.2.2 Client lourd . . . 13
4.3 Intégration des Web services . . . 15
4.3.1 Côté serveur . . . 15
4.3.2 Côté client . . . 15
4.3.3 Gestion des exceptions . . . 17
4.3.4 Tests unitaires . . . 18
4.4 Plugin serveur . . . 19
4.4.1 Administration . . . 19
4.4.2 Vérification . . . 20
4.4.3 Classes utilitaires . . . 21
4.4.4 Politique de gestion des communautés instantanées . . . 22

4.4.5 Internationalisation de la console 22
4.4.6 Problèmes rencontrés 23
4.5 Répartition des tâches 23

5 Contenu du livrable 25
5.1 Répertoire client 27
5.2 Répertoire Web_Services 27
5.3 Le répertoire plugin 28
5.4 Le répertoire doc 28
5.5 Le répertoire lib 28

6 Synthèse 29
6.1 Bilan technique 29
6.1.1 Bilan fonctionnel 29
6.1.2 Bilan non fonctionnel 30
6.1.3 Vérification et Validation 30
6.2 Bilan personnel 30

A Définitions et acronymes 33
A.1 Web Services 33
A.2 Protocole SOAP 33
A.3 WSDL 33
A.4 AXIS 34
A.5 WSDD 34
A.6 Wildfire 34
A.7 Spark 34
A.8 Protocole XMPP 34
A.9 Jabber 34
A.10 IETF 35
A.11 Client XMPP 35
A.12 PDA 35
A.13 Imov 35
A.14 Mysaifu 35
A.15 Apache Tomcat 35

B Clients XMPP testés 37
B.1 Clients XMPP fonctionnant sur WinCE 37
B.2 Clients Java XMPP 37
B.3 Clients XMPP sur navigateurs internet 37
B.4 Clients XMPP fonctionnant sur une J2ME 37

Chapitre 1

Introduction

1.1 Objectifs

Pour faire face à l'évolution de leur métier, les entreprises doivent pouvoir intégrer anciens et nouveaux systèmes d'information par composition. Cette intégration par composition de services est actuellement pour elles un enjeu crucial. La technologie des *web services* a justement été conçue pour faciliter *l'interopérabilité des applications* en permettant les invocations de services quelconques au travers du réseau. Parallèlement, de nombreuses plates-formes communautaires existent, formées sur le modèle de MSN Messenger et de ses extensions. Mais elles ne permettent pas de *former des groupes de façon instantanée* alors qu'un certain nombre d'applications sont envisageables.

Le but de ce projet baptisé **Amui**[1], est de fournir une application qui gère des *communautés instantanées en pilotant d'autres applications à l'aide de web services*. En utilisant un client spécifique, les utilisateurs se connectent au serveur et sont automatiquement placés dans des groupes d'intérêts en fonction des *mots-clés* qui les caractérisent. Chaque communauté instantanée ainsi formée sera configurée pour exécuter différentes applications (messagerie instantanée, diffusion audio ou vidéo, etc.). Chaque application est intégrée coté client et serveur par un web service, ce qui facilite l'intégration et le pilotage de nouveaux services dans la plate-forme. Le projet développera une plate-forme serveur sous Linux et/ou Windows, et des clients sous Linux, Windows et Pocket Pc.

1.2 Principes

Prenons l'exemple du forum de l'emploi de Nice. Un serveur de communauté instantanée a été mis à disposition pour faciliter la visite.
Un utilisateur cherchant du travail dans le domaine de l'informatique entre dans son client de communauté instantanée (application java) les mots clés représentatifs de ses centres d'intérêt : informatique,recherche et emploi. Il est automatiquement affecté au salon réservé à la recherche d'emploi dans l'informatique.
En utilisant ensuite un client XMPP de messagerie instantanée, il est de suite mis en contact avec les autres membres du salon étant déjà connectés, dans notre exemple : IBM, SUN, mais aussi des personnes à la recherche d'emploi dans l'informatique.

[1] Amui signifie se grouper en tahitien

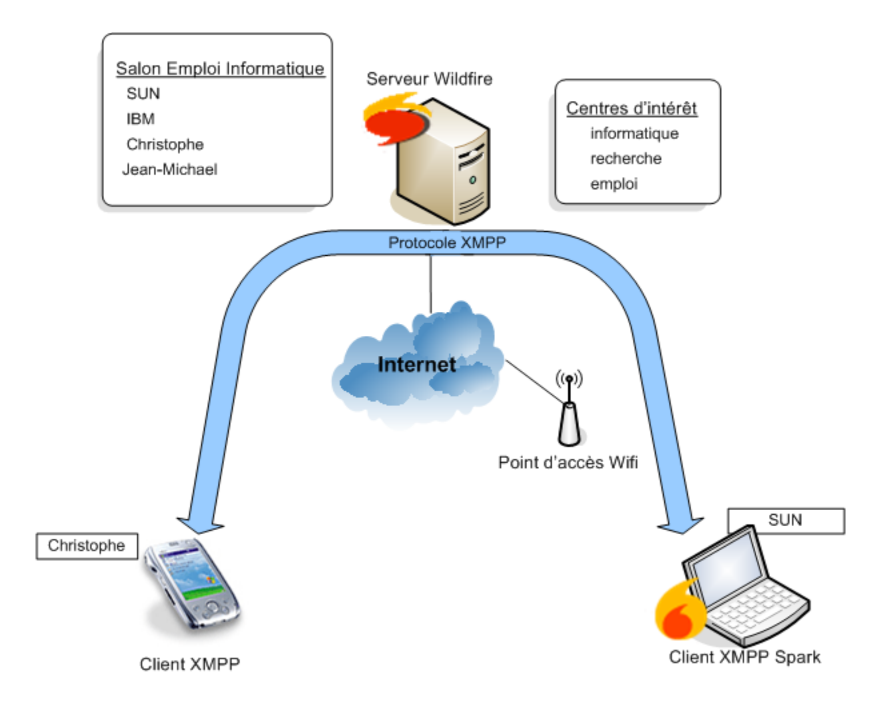


Fig. 1.1 – Deux clients ayant les mêmes centres d'intérêts peuvent être mis en relation grâce au client de communautés instantanées

1.3 Architecture générale

Vous pouvez observer sur le schéma qui suit (schéma 1.2) l'architecture générale de la version finale de notre projet (avec client lourd).

Nous reprenons des technologies déjà existantes comme Wildfire, Axis et Tomcat (représentées en bleu) et nous y ajoutons des fonctionnalités (représentées en jaune).

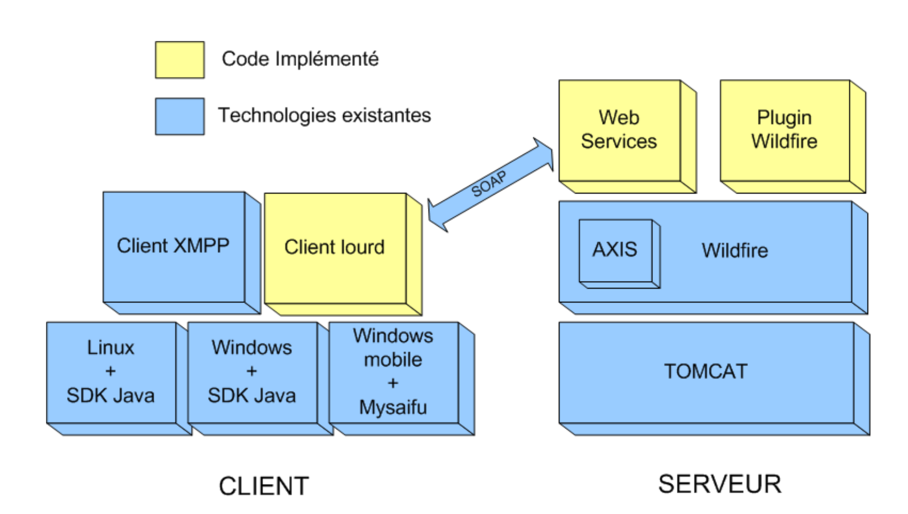


FIG. 1.2 – Architecture de notre application

Chapitre 2

Cahier des charges

Le cahier des charges suivant est celui qui à été établi avant la présoutenance.

2.1 Connectivité client/serveur

2.1.1 Création d'un couple compte/mot de passe

Client léger L'utilisateur doit utiliser son navigateur pour charger la page web qui lui permet :
- de choisir ses mots clés, dans la première version du client, il s'agira d'une liste de mots à sélectionner, dans la seconde, l'utilisateur devra effectuer une "recherche" des salons par mots clé.
- de proposer un login afin de créer un compte et de se connecter au serveur Wildfire par l'intermédiaire de son client XMPP. Si le login est déjà assigné à une personne, l'application (plugin) lui en proposera un autre. Une fois le login accepté, l'application donne un mot de passe qui ne sera valide que le temps de la connexion.

Client lourd L'utilisateur doit installer sur sa machine une application java qui lui permettra de sélectionner/saisir ses mots clés. Contrairement au client léger, le client lourd sauvegarde localement les mots clés et le login proposés à la session précédente. La fonction du client reste la même, son rôle est de configurer le compte de l'utilisateur avant chaque connexion.

Connexion/Déconnexion Le client XMPP doit permettre à une personne, ayant obtenu un login et un mot de passe, de se connecter au serveur Wildfire et d'accéder aux salons de discussion auxquels elle a été assignée. La déconnexion se fait également à partir du client XMPP.

2.1.2 Sauvegarde des mots clés pour le client lourd

Le client lourd doit sauvegarder les données du client (mot clés, login) dans un fichier.

2.2 Création dynamique des communautés instantanées

2.2.1 Analyse des mots clés

Pour faciliter la recherche de salons, le plugin analyse les mots clés entrés par l'utilisateur et propose les salons dont l'orthographe des noms s'y approche.

2.2.2 Affectation aux salons

Le plugin permet d'indiquer au serveur Wildfire à quels salons affecter les utilisateur en fonction des mots clés sélectionnés.

2.2.3 Notification

Lorsqu'un utilisateur entre ou sort d'un salon, sa présence/absence est notifiée auprès des autres utilisateurs du salon.

2.3 Administration

2.3.1 Plugin ou module pour Wildfire

Nous utiliserons un plugin si celui-ci est capable de communiquer directement avec les web services, sans avoir à passer par une API. Le cas échant, nous développerons un module. Le module sous Wildfire est prévu pour communiquer avec des web services.

Dans tous les cas, que nous utilisions un module ou un plugin, celui-ci devra :
- vérifier qu'un login est libre et en proposer un nouveau sinon
- créer les mots de passe
- créer les comptes
- assigner les utilisateurs aux salons

2.3.2 Console d'administration

Une console d'administration du plugin intégrée à celle de Wildfire permettra de configurer le plugin/module (et de surveiller son bon fonctionnement).

2.3.3 Console de test

Les tests JUnit ne pouvant être effectués car nous développons un plugin. Une console de test permettra, à toute personne voulant utiliser notre plugin, de vérifier qu'il fonctionne correctement. Celle-ci se présentera sous la forme d'une page web avec un bouton qui permettra de lancer plusieurs séries d'opérations d'administration des communautés, telles que la création/suppression d'utilisateurs et de salons, affectation à plusieurs salons etc. La console générera un fichier de logs que l'administrateur devra comparer avec le notre. Si les fichiers sont les mêmes, le plugin fonctionne correctement

Chapitre 3

Planning du projet

Nous avons globalement respecté le planning que nous nous étions fixé lors de la phase de gestion de projet.

Nous avons juste dû décaler la remise des lots car nous les avions fixés lors de jours feriés. Nous avons donc accumulé une journée de retard par semaine. Cependant nous avions prévu la semaine du 24 au 30 avril pour pallier à d'éventuels problèmes de planning. Cela n'a donc pas eu d'incidence sur l'ensemble du projet.

3.1 Client

Prévisionnel

ID	Nom	Début	Fin
8	Client	10/04/2006	23/04/2006
8-1	Page Web (Client léger)	10/04/2006	16/04/2006
8-2	Application Java (Client lourd)	10/04/2006	23/04/2006
8-2-1	Cliend lourd	17/04/2006	23/04/2006
8-2-2	Web Services	17/04/2006	23/04/2006
8-3	Tests et Documentations	10/04/2006	23/04/2006
9	Fournitures	17/04/2006	24/04/2006
9-1	1er Lot	17/04/2006	17/04/2006
9-2	2ème Lot	24/04/2006	24/04/2006
10	Prise en compte des retours	24/04/2006	30/04/2006
10-1	Prise en compte des retours	24/04/2006	30/04/2006

avril 2006 mai 2006
10 17 24 01 08 15

Effectif

ID	Nom	Début	Fin
8	Client	10/04/2006	03/05/2006
8-1	Base	10/04/2006	17/04/2006
8-1-1	Interface Web	10/04/2006	17/04/2006
8-1-2	Web services	10/04/2006	17/04/2006
8-2	Améliorations	19/04/2006	03/05/2006
8-2-1	Client lourd	19/04/2006	03/05/2006
8-2-2	Améliorations du client léger	24/04/2006	03/05/2006
8-2-3	Améliorations des web services	19/04/2006	25/04/2006
8-2-4	Tests unitaires	26/04/2006	03/05/2006
9	Fournitures	18/04/2006	09/05/2006
9-1	1er lot	18/04/2006	18/04/2006
9-2	2ème lot	26/04/2006	26/04/2006
9-3	3ème lot	09/05/2006	09/05/2006

avril 2006 mai 2006
10 17 24 01 08 15

Base du client (Interface web) — Améliorations du client (Client lourd) — Prise en compte des retours (améliorations) — Remises des lots

FIG. 3.1 – Planning prévisionnel et effectif concernant la partie client

Interface web A l'issue de la remise du 1er lot, nous avons rajouté une fonctionnalité au plugin : la possiblité de choisir son mot de passe (avant cela il était généré automatiquement). Pour ce faire, l'implémentation du plugin a été modifié. Nous avons donc du (du 24 avril au 3 mai) revoir l'interface web pour quelle permette d'utiliser ces nouvelles fonctionalités. Ceci n'était pas prévu dans notre planning initial mais nous avons pu nous en occuper parallement aux autres tâches prévues durant cette période.

Web services sur Pocket PC En ce qui concerne les web services côté client, nous avions prévu d'utiliser Axis aussi bien sur une plateforme classique (Windows, Linux) que sur PDA mais nous avons été obligés de procéder differement en ce qui concerne le Pocket PC. Après 3 jours de travail nous avons décidé avec l'accord de nos encadrants d'utiliser une autre alternative : **KSOAP** afin de ne pas accumuler de retard. (cf. 4.3.2)

Client XMPP sur Pocket PC Lors de la phase de gestion, nous avions trouvé un client XMPP fonctionnant sur Pocket PC : **imov**. Mais nous nous sommes rendu compte par la suite

qu'il ne supportait pas les salons de discussion (au début nous avions prévu d'utiliser des groupes de discussion et non des salons de discussion cf. 4.4.4.1). Nous avons donc commencé à chercher un nouveau client XMPP pour PDA supportant les salons mais malheureusement nous n'en avons trouvé aucun (voir Annexe B). Cette tâche a dû se faire parallèlement aux autres tâches prévues. Mais cela n'a pas entraîné de retard dans le planning.

3.2 Serveur

Prévisionnel

ID	Nom	Début	Fin
7	**Serveur**	**10/04/2006**	**20/04/2006**
7-1	**Base**	**10/04/2006**	**16/04/2006**
7-1-1	Plugin	10/04/2006	16/04/2006
7-1-2	Web Services	10/04/2006	16/04/2006
7-2	**Améliorations**	**10/04/2006**	**20/04/2006**
7-2-1	Améliorations	17/04/2006	20/04/2006
7-3	Tests et Documentation	10/04/2006	20/04/2006
9	**Fournitures**	**17/04/2006**	**24/04/2006**
9-1	1er Lot	17/04/2006	17/04/2006
9-2	2ème Lot	24/04/2006	24/04/2006
10	**Prise en compte des retours**	**24/04/2006**	**30/04/2006**
10-1	Prise en compte des retours	24/04/2006	30/04/2006

Effectif

ID	Nom	Début	Fin
7	**Serveur**	**10/04/2006**	**03/05/2006**
7-1	**Base**	**10/04/2006**	**17/04/2006**
7-1-1	Classes métiers	10/04/2006	17/04/2006
7-1-2	Console d'administration	10/04/2006	17/04/2006
7-1-3	Web Services	10/04/2006	17/04/2006
7-2	**Améliorations**	**19/04/2006**	**03/05/2006**
7-2-1	Ajout de fonctionnalités	19/04/2006	25/04/2006
7-2-2	Console de tests	19/04/2006	03/05/2006
7-2-3	Améliorations de la console d'admin	19/04/2006	03/05/2006
9	**Fournitures**	**18/04/2006**	**09/05/2006**
9-1	1er lot	18/04/2006	18/04/2006
9-2	2ème lot	26/04/2006	26/04/2006
9-3	3ème lot	09/05/2006	09/05/2006

FIG. 3.2 – Planning prévisionnel et effectif concernant la partie serveur

Nous avons étudié différentes manières d'utiliser les web services :
- en utilisant Axis et Wildire comme deux applications web indépendantes : Nous n'avons pas réussi à faire communiquer Axis avec notre plugin dans Wildfire
- en intégrant Axis directement au plugin : Après de nombreux posts sur les forums de Jive-software, il est apparu que cette solution nous aurait pris beaucoup de temps, sans aucune garantie d'obtenir un résultat.
- en intégrant Axis dans wildfire : Cette solution nous est apparue comme étant la meilleure étant donné le temps qui nous était imparti.

Nous avons effectué en parallèle des recherches dans le sens des deux premières solutions avant de changer d'architecture pour adopter la troisième. Ce changement nous a pris une journée.

3.3 Documentation et préparation à la soutenance

Prévisionnel

ID	Nom	Début	Fin
7	**Serveur**	**10/04/2006**	**20/04/2006**
7-3	Tests et Documentation	10/04/2006	20/04/2006
8	**Client**	**10/04/2006**	**23/04/2006**
8-3	Tests et Documentation	10/04/2006	23/04/2006
11	**Documentation**	**01/05/2006**	**07/05/2006**
11-1	Aide embarquée (utilisateur)	01/05/2006	07/05/2006
11-2	Aide HTML (programmeur)	01/05/2006	07/05/2006
12	**Soutenance**	**03/05/2006**	**10/05/2006**
12-1	Rapport	03/05/2006	09/05/2006
12-2	Support	03/05/2006	09/05/2006
12-3	Soutenance	10/05/2006	10/05/2006

Effectif

ID	Nom	Début	Fin
10	**Documentation**	**16/04/2006**	**15/05/2006**
10-1	Manuel d'installation	16/04/2006	09/05/2006
10-2	Manuel de conception	16/04/2006	09/05/2006
10-3	Choix d'implémentation	03/05/2006	09/05/2006
11	**Soutenance**	**02/05/2006**	**15/05/2006**
11-1	Rapport	02/05/2006	09/05/2006
11-2	Préparation de la soutenance	09/05/2006	14/05/2006
11-3	Soutenance	15/05/2006	15/05/2006

FIG. 3.3 – Palnning prévisionnel et effectif concernant la documentation et la préparation de la soutenance

3.3.1 Rédaction de la documentation

Nous prévoyions de rédiger une aide utilisateur pour le client lourd. Mais lors de son développement nous avons trouvé qu'il n'était pas vraiment nécessaire. Nous avons préféré nous concentrer sur une bonne ergonomie.

Mais durant cette même phase, nous nous sommes rendu compte que l'installation du serveur était plutôt longue et compliquée. Nous avons donc décidé de mettre à profit le temps réservé à la documentation utilisateur du client lourd pour rédiger un manuel d'installation. Ceci n'a donc pas eu d'impact sur notre planning.

3.3.2 Préparation de la soutenance

La date de la soutenance ayant été décalée au 15 mai, nous pouvions préparer notre soutenance finale du 9 au 14 mai. Ceci nous a donc permis de nous concentrer essentiellement sur la rédaction du rapport du 2 au 9 mai.

C'était, pour toute l'équipe, la première fois que nous concevions un planning de projet avec autant de rigueur. Nous nous rendons compte que cela nous a beaucoup facilité la tâche lors de la phase de développement car cela nous fixait des limites et nous donnait un rythme de travail. Nous en retenons que la phase de gestion est une étape indispensable à la réalisation d'une application.

Chapitre 4

Travail réalisé

4.1 Architecture logicielle

4.1.1 Architecture générale

Vous pouvez observer sur le schéma ci-dessous (schéma 4.1) l'architecture générale du 1er lot (à gauche) et celle de la version finale (à droite).

Le schéma de gauche montre la possibilité de déployer le client léger sur un autre serveur que celui de wildfire (cf. 4.2.1).

Celui de droite montre que l'application Java communique directement avec le plugin par web services.

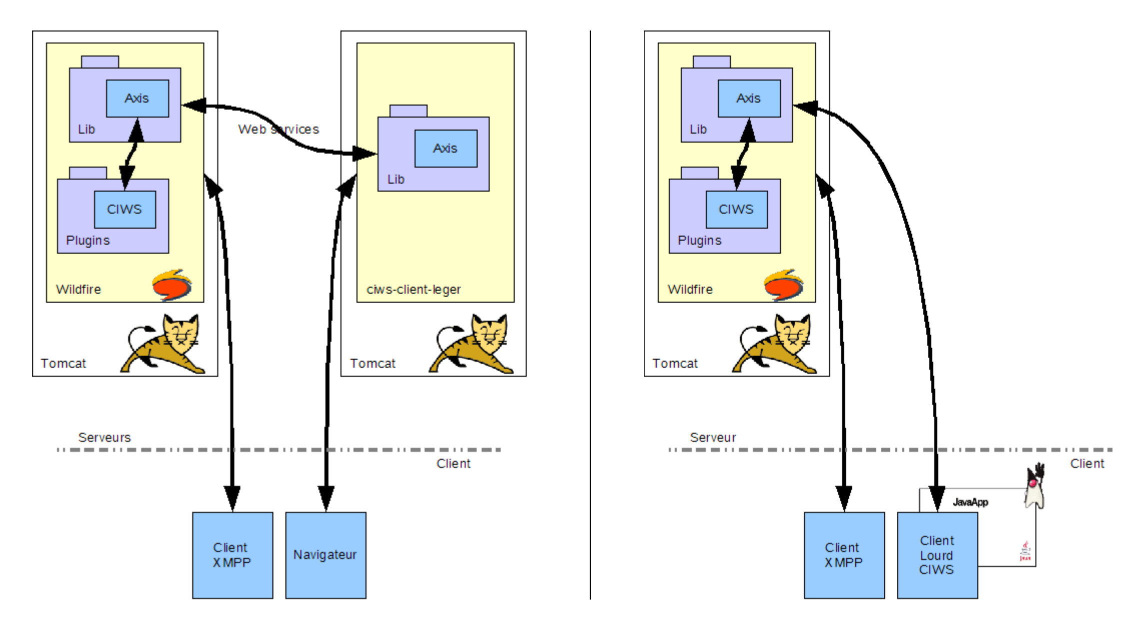


Fig. 4.1 – A gauche l'architecture du 1er lot, et à droite celle du 2ème

4.1.2 Déploiement sur Tomcat des applications Axis et Wildfire

L'application Amui doit permettre de communiquer via web services avec un plugin Wildfire. Le déploiement d'Axis et Wildfire en deux applications web semblait être la façon la plus naturelle. Mais cela posa un problème de communication entre ces deux applications, car justement nous voulions utiliser Wildfire par web services.

Nous avons finalement choisi une autre approche. Intégrer Axis dans Wildfire. Ceci permet à l'implémentation des web services de communiquer avec le serveur Wildfire et le plugin.

Voici le schéma des deux modèles évoqués précédemment :

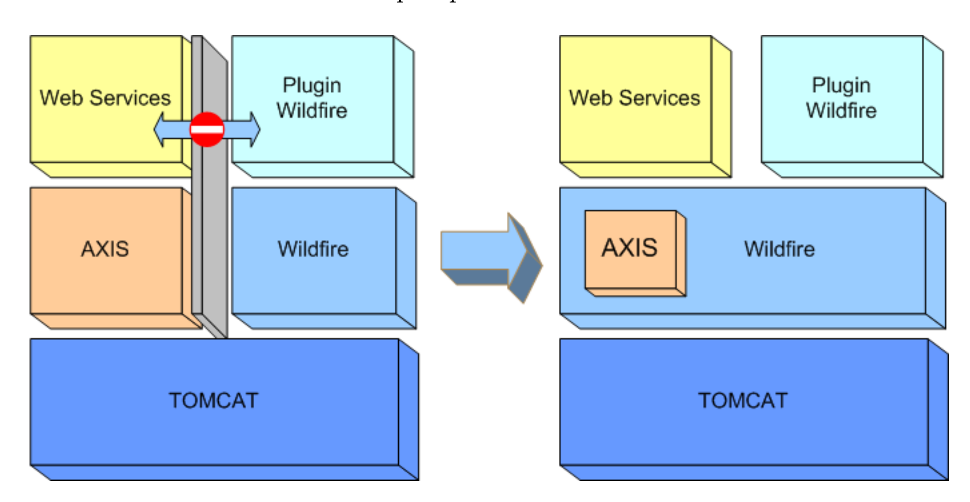


Fig. 4.2 – Changement de modèle

4.2 Client

Nous voulions pouvoir faire communiquer le plus tôt possible le client et le serveur via web services. C'est pourquoi nous avons décidé de commencer par développer un client léger (interface web) avant de réaliser un client lourd (application java). Ceci était plus facile car les web services étaient entièrement sur le serveur et nous n'avions donc pas de problème avec le cas particulier du PDA.

C'est pourquoi l'application Amui offre deux clients :
- Une application web : client léger
- Une application Java : client lourd

Ces deux clients ont été implémentés par Michèle Reynier, et Jean-Michael Legait.

4.2.1 Client léger

- Fonctionnalités de base : du 10 à 17 Avril
- Améliorations : du 24 Avril au 3 Mai

Le client léger est une application web. Elle s'appuie sur une Servlet et deux JSP. Ce client peut très bien être déployé sur un serveur différent de celui qui héberge le serveur de messagerie Wildfire. Voir schéma 4.1 partie de gauche

Voici les fonctionalités de base implémentés par ce client léger :
- Saisir son login et son mot de passe
- Indiquer si l'on a déjà un compte ou si l'on est un nouvel utilisateur
- Choisir des salons parmis ceux qui sont proposés

JSP/Servlet : Voici le fonctionnement utilisé dans le client léger :

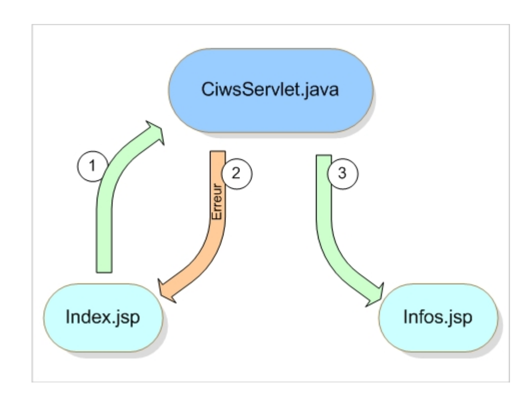


Fig. 4.3 – Interaction JSP/Servlet

- (1) Envoi des données saisies.
- (2) En cas de login déjà pris, ou de mot de passe incorrect, l'utilisateur est invité à recommencer.
- (3) Envoi des données pour le récapitulatif.

4.2.2 Client lourd

- Fonctionnalités de base : du 19 au 24 Avril
- Améliorations : du 25 Avril au 3 Mai

4.2.2.1 Interface graphique

Le client lourd est une application avec une interface graphique AWT. Nous utilisons AWT et non SWING, car cette bibliothèque est beaucoup moins gourmande en ressource. Et ne perdons pas de vu que ce client lourd doit fonctionner aussi bien sur ordinateur de bureau, que sur Pocket PC.

De plus, AWT plus ancien que SWING, est mieux reconnu par les différentes plateforme.

Problème d'affichage sur Pocket PC Nous avons voulu insérer des images dans l'interface graphique du client. Mais nous n'obtenions pas le même résultat en fonction de la plate-forme matérielle utilisée. En effet sur un PC les images s'affichaient, alors que sur un Pocket PC aucune image n'apparaissait. Nous avons après de nombreuses recherches trouvé la cause, et la solution au problème.

Lorsque nous construisons un objet de la classe Image, le téléchargement du fichier image se fait en tâche de fond. L'objet obtenu n'est donc pas immédiatement opérationnel, et nous nous retrouvons à manipuler non pas une image mais un proxy.

Nous avons donc utiliser un "MediaTracker". La classe MediaTracker permet de demander l'affichage de l'image, une fois le téléchargement terminé.

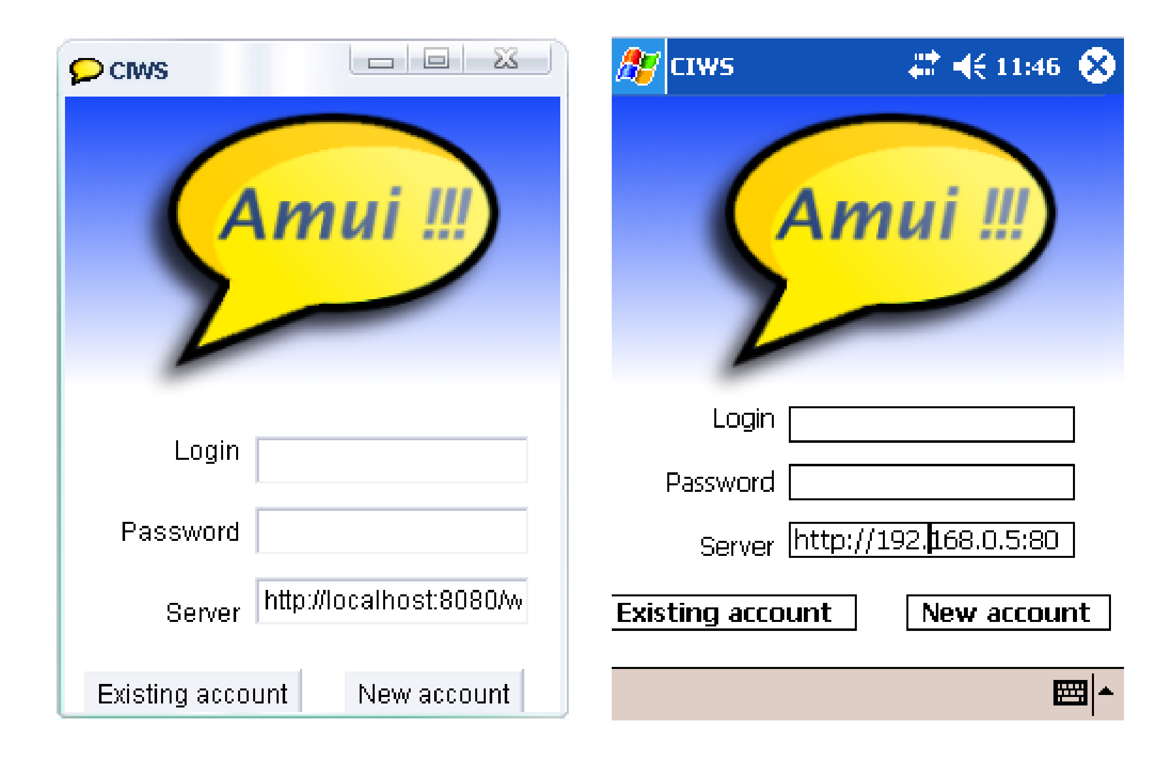


FIG. 4.4 – Page d'accueil du client lourd sous Windows (PC) et Windows Mobile (Pocket PC)

4.2.2.2 Sauvegarde des mots clés

Nous voulions sauvegarder les mots clés de l'utilisateur dans un fichier XML (format très répandu, et très pratique). Nous utilisions la bibliotèque DOM pour manipuler le fichier de sauvegarde. Mais la JVM du Pocket PC que nous utilisons ne dispose pas de cette bibliothèque. Nous avons donc choisi un second format : un fichier de propriété ".properties" et décidé de fournir deux modes de sauvegarde en fonction de la plate-forme matérielle utilisée.

Le client Amui possède donc deux formats de sauvegarde des mots clés.
- Dans un fichier XML pour PC (*save.xml*)
- Dans un fichier ".properties" pour Pocket PC (*save.properties*)

4.3 Intégration des Web services

4.3.1 Côté serveur

L'intégration de Web services comme intermédiaire entre le client et le serveur de la messagerie instantanée garantit une utilisation indépendante des plate-formes du client. Le Web service du serveur, nommé Ws2Wf (Web service To WildFire), permet donc de communiquer avec le plugin Wildfire, à partir d'un client. Ce Web service propose toutes les methodes nécessaires pour que le serveur de la messagerie instantanée puisse gérer le client (ajouter le client, l'affecter à un ou plusieurs salons...).

Nous avons utilisé Apache Axis afin d'implémenter notre Web Service. Enfin le déploiement du Web Service s'effectue à l'aide d'un fichier de deploiement spécifique à notre Web service qui est le fichier WSDD.

4.3.2 Côté client

Nous avons développé deux types de client : le client léger et le client lourd. Le client léger est une page Web auquel l'utilisateur peut accéder simplement avec son navigateur. La communication entre le client et le serveur est basée sur le protocole HTTP. Puis la Servlet du client léger effectue les appels Web Services (protocole SOAP). Le client léger n'a donc pas d'interaction directe avec les Web Services.

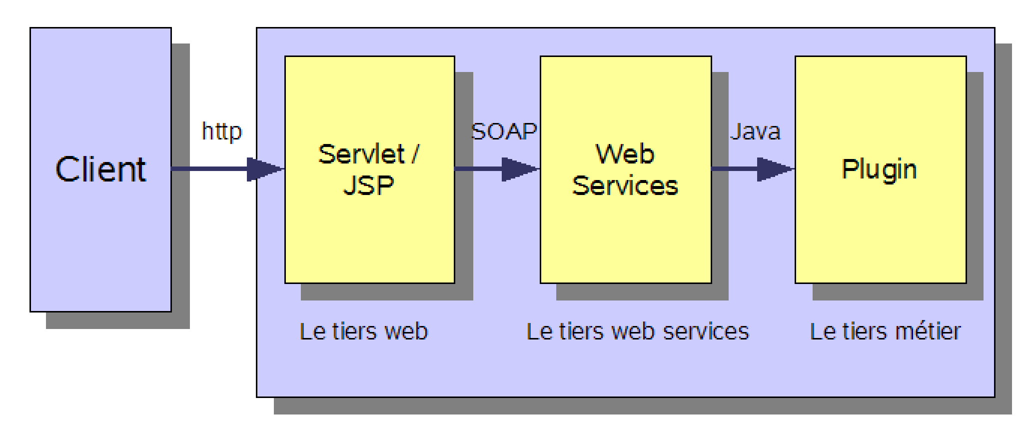


FIG. 4.5 – Architecture Client léger

La seconde grande étape fut le développement d'un client lourd (Application Java). L'utilisateur n'utilise donc plus son navigateur pour accéder aux Web Services mais une application Java. Le client effectue alors directement les appels aux Web Services grâce au protocole SOAP.

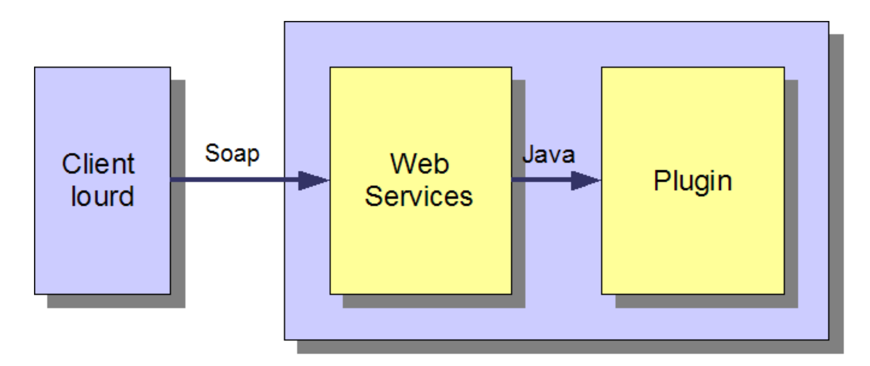


Fig. 4.6 – Architecture Client lourd

Lors de cette étape nous avons constaté qu'il était très difficile d'implémenter de la même manière les web services (en utilisant Axis) sur une plate-forme pc que sur un Pocket-PC. En effet, sur les PC, nous utilisons la bibliothèque d'Apache Axis afin d'effectuer les appels aux Web Services.
Cependant, nous avons constaté qu'Axis possède des dépendances vers d'autres bibliothèques qui sont supposées se trouver dans une JVM standard PC (J2SE). Bien que nous ayons utilisé une JVM "compatible" J2SE sur le PDA (Mysaifu), "certifiée" GnuClasspath[1], nous avons constaté que cela ne suffisait pas. En effet, Apache Axis utilise aussi certaines bibliothèques supplémentaires à GnuClasspath, dites GnuClasspathX.
Malheureusement, notre JVM Mysaifu ne possède pas ces bibliothèques. Après de nombreuses tentatives d'intégration des bibliothèques absentes (difficilement identifiables), nous avons décidé d'implémenter le client du Pocket-PC sans utiliser Apache Axis. Nous avons finalement utilisé une bibliothèque plus "légère" et plus adaptée à une JVM limitée : KSOAP. KSOAP, tout comme Axis, nous permet de construire notre requête SOAP et de la faire parvenir au Web Service.

Etant donné que nous avons plusieurs implémentations du client en ce qui concerne les appels aux Web Services, il était nécessaire de mettre au point un patron de conception (Design Pattern Factory) afin d'abstraire les classes utilisées.
Pour faciliter le paramétrage de l'application, nous avons également utilisé un fichier de propriétés contenant des informations telles que l'adresse du serveur, le type de web service à utiliser, etc.

[1] qui possède les librairies standards d'une JVM classique

4.3.2.1 Client PC

Nous avons implémenté deux types de client PC. En effet, Apache Axis nous propose deux manières d'implémenter un client Web Service. Soit en fabriquant nous même la requête SOAP, et dans ce cas préciser les informations nécessaires : les types utilisés (entier, flottant...), les méthodes à invoquer, l'adresse du Web Service... Soit en utilisant les proxies (stubs) qui sont générés par l'outil WSDL2Java, qui contiennent la plupart des informations utiles. Le code du client devient alors extrêmement simple. Notre première approche a été le modèle sans proxies car ce dernier est plus facile à comprendre et donc plus rapide à implémenter. Ceci nous a permis de vérifier rapidement la validité de la chaîne de fonctionnement. C'est ensuite que nous avons implémenté le modèle avec proxies qui rend le code du client plus court et plus simple.

4.3.2.2 Client PDA

En ce qui concerne le Pocket-PC, nous avons utilisé KSOAP, qui se rapproche du premier type de client PC car il nous est nécessaire de fabriquer nous même la rêquete SOAP. Cependant KSOAP, plus adapté à une JVM limitée, nous assure un meilleur fonctionnement.

4.3.3 Gestion des exceptions

Une autre difficulté rencontrée avec les Web Services a été la gestion des exceptions Java, notamment lors de leur passage à travers le réseau. En effet, une exception envoyée par un Web Service est transformée en une exception particulière (SOAP FAULT CODE) dans laquelle est encapsulée l'exception initiale. Or, notre implémentation du client nous impose d'utiliser cette exception.
Nous avons donc décidé qu'une exception serait traduite en un code de retour, qui passera à travers le réseau, et sera de nouveau traduit en exception du côté du client.

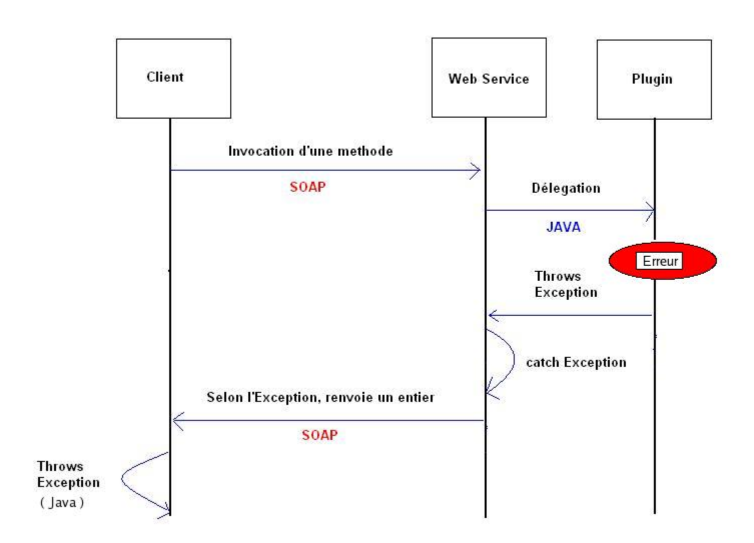


Fig. 4.7 – Gestion des Exceptions

4.3.4 Tests unitaires

Nous avons testé le bon fonctionnement de nos implémentations de client Web Services à l'aide de JUnit 4.0 (des tests unitaires). Nous avons choisi la version 4.0 de JUnit car elle est la seule à pouvoir gérer les exceptions (vérifier que la bonne exception a été lancée par exemple.)
Les principaux tests unitaires effectués permettent de vérifier :

- l'ajout d'un nouvel utilisateur (pseudonyme, mot de passe, liste des mots clés) au serveur, en supposant que le pseudonyme n'est pas déjà utilisé.
- la connexion d'un utilisateur existant au serveur (en supposant qu'il existe déjà).
- l'ajout d'un nouvel utilisateur (pseudonyme, mot de passe, liste de mots clés) au serveur, en supposant que le pseudonyme est déjà utilisé.
- la connexion d'un utilisateur existant au serveur avec un mot de passe erroné (en supposant qu'il existe déjà).

4.4 Plugin serveur

Le plugin a été implémenté par Michèle BARRE et Christophe ROGER

4.4.1 Administration

4.4.1.1 Console web

Son rôle est de proposer toutes les fonctionnalités de base pour gérer les communautés instantanées en étendant le mécanisme d'administration web proposé par wildfire. La console web permet :

- La visualisation des utilisateurs et communautés.
- La création, la modification et la création des communautés et des utilisateurs.
- L'assignation des utilisateurs aux communautés.
- La modification des propriétés du serveur de communautés qui sont contenues dans un fichier de propriétés.

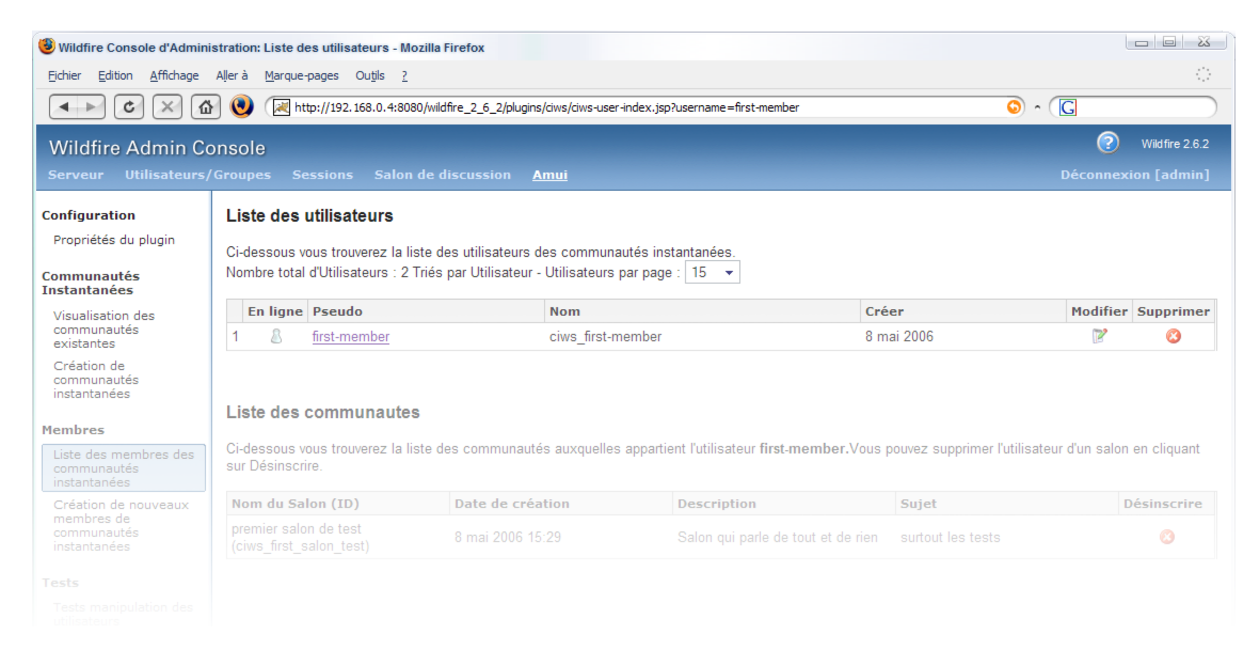


FIG. 4.8 – Console de visualisation des utilisateurs

4.4.1.2 Classes métiers

1. Interface Plugin
   Notre classe AmuiPlugin doit obligatoirement implémenter cette interface afin d'être vue en tant que plugin par wildfire. Elle doit implémenter deux méthodes, *initializePlugin* et *destroyPlugin* qui vont définir les actions à accomplir au chargement et à la suppression du plugin.
2. Interface InstantCommunities
   Cette interface met à disposition 3 méthodes qui sont utilisées par les web services :
   - *addUserToRoom(String,String,String,Boolean)* : ajoute un utilisateur dans les salons correspondant aux mots clés. Si l'utilisateur n'existe pas celui-ci est créé.
   - *getRooms()* : rend le nom de tous les salons disponibles sur le serveur.
   - *getRooms(String)* : rend les salons auxquels l'utilisateur est inscrit.
3. Interface InstantCommunitiesAdmin
   Toute console d'administration devant gérer les communautés instantanées doit implémenter cette interface. Elle permet à la console d'utiliser les fonctions d'administration de base du plugin :
   - *createRoom(String, String, String, String)* Création et modification de communauté
   - *destroyRoom(String)* Suppression de communauté
   - *changeRooms(String[], String)* Modification de l'affectation d'un utilisateur aux communautés.
   - *createMember(String, String)* Création d'un utilisateur de communauté instantanée

4.4.2 Vérification

4.4.2.1 Console web

Le rôle de la console de test est de vérifier le bon fonctionnement du plugin en lançant des tests unitaires et en en visualisant les résultats. L'intégration du plugin au sein du serveur Wildfire rend difficile la réalisation de tests JUnit. Pour permettre un contrôle de qualité nous avons donc mis en place des fichiers de logs, et une console de visualisation des logs dans les pages de la console de test. Cette console enregistre toute l'activité du plugin et a été développé pour permettre son utilisation avec de nouvelles classes de tests sans aucune modification de la page. Pour l'instant, les tests vérifient les points suivants :

- le fonctionnement des méthodes de création et de suppression des utilisateurs de communautés instantanées
- le fonctionnement des méthodes de création, de modification et de suppression des communautés instantanées
- le fonctionnement des méthodes d'affectation des utilisateurs aux communautés, de réaffectation à de nouvelles communautés, et de désabonnement à des communautés en fonction des mots clés.

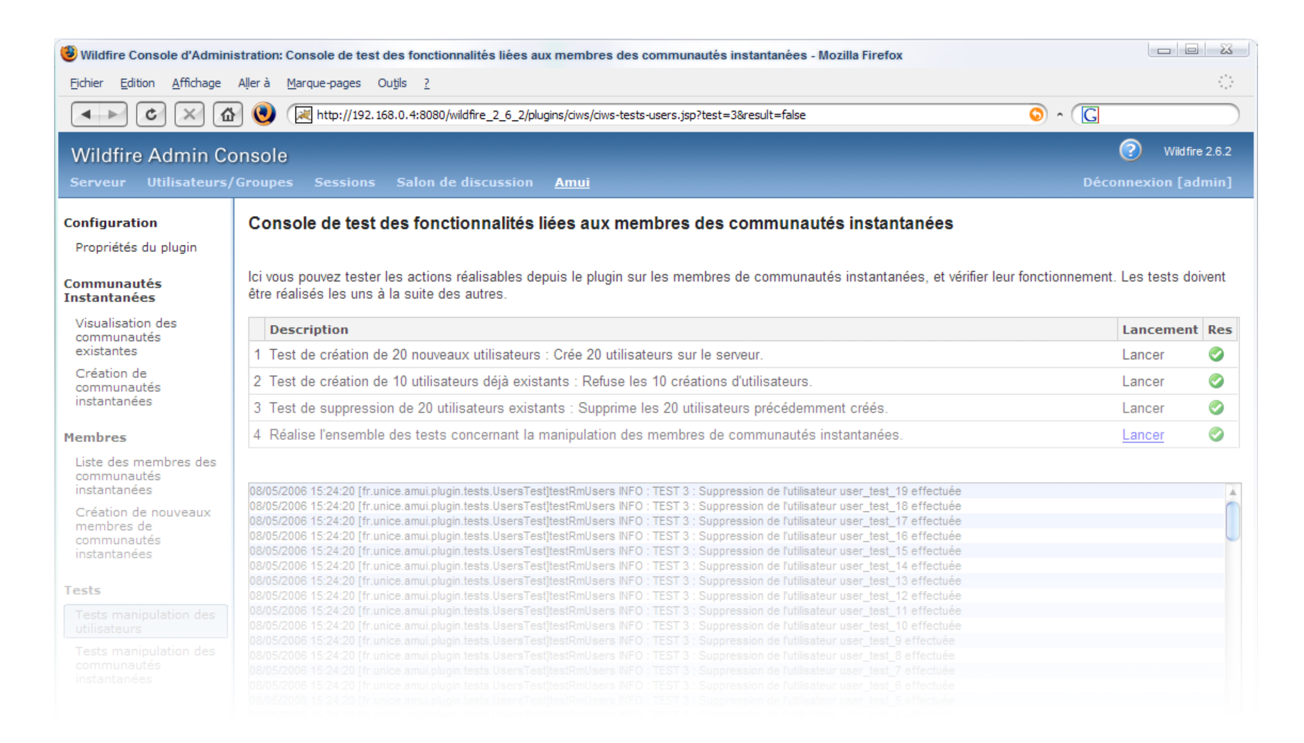


FIG. 4.9 – Console de test des communautés instantanées

4.4.2.2 Classes de test

Les classes de tests doivent implémenter une interface commune : *fr.unice.amui.plugin.tests.Test*. Cette interface définit ce que doit faire un test à travers les méthodes suivantes :

- *init()* : initialise l'environnement pour pouvoir exécuter le test
- *run(int i)* : permet d'effectuer le ième sous-test.
- *run()* : lance le test dans sa globalité, exécutant les sous-tests les uns après les autres
- *close()* : permet de remettre l'environnement dans l'état qu'il avait avant l'exécution des tests

L'utilisation de cette interface permet d'abstraire l'utilisation des tests. Nous pouvons désormais utiliser une fabrique et ainsi favoriser la réutilisation du code. Grâce à cette architecture, la page jsp utilisée pour la console de tests reste valable pour les prochaines implémentations de l'interface Test. Seule la fabrique nécessite alors d'être mise à jour.

4.4.3 Classes utilitaires

4.4.3.1 Analyse des mots clés

Toute classe responsable de la recherche des salons par analyse des mots clés doit impérativement implémenter l'interface InstantCommunitiesSearch. Celle-ci contient 3 méthodes :

- *index(String,String)* : ajoute une communauté et ses informations dans l'index.
- *update()* : recrée l'index.
- *search(String)* : donne la liste des salons correspondant aux mots clés.

La classe AmuiIndexer utilise l'api d'Apache Lucene et permet de créer un fichier en mémoire dans lequel sont indexés des documents. Ces documents sont constitués de l'identifiant de chaque communauté et des informations relatives à ces communautés (description, sujet, nom de la communauté).

L'utilisation de Lucene nous permet d'affecter l'utilisateur aux salons adéquats en faisant une recherche par proximité des mots clés dans les informations des communautés qui ont été indexées.

4.4.3.2 Gestion des logs

Pour pourvoir utiliser au mieux les logs, nous avons décidé d'implémenter un «Handler» personnalisé qui va enregistrer dans une variable les différents logs pour pouvoir les afficher sur la console de test.

4.4.3.3 Exceptions

Deux classes d'exceptions ont été implémentées :
- *WrongLoginOrPasswordException* est lancée si le couple login/mot de passe n'est pas valide.
- *LoginAlreadyExistsException* est lancée lorsque qu'un utilisateur désire créer un compte avec un login déjà utilisé.

4.4.4 Politique de gestion des communautés instantanées

4.4.4.1 Utilisation de salons

Nous avions initialement prévu d'utiliser la notion de groupe pour représenter nos communautés, cependant après avoir étudié les possibilités de Wildfire nous nous sommes rendu-compte que la notion de salon était plus appropriée. En effet, avec les salons, le principe de conversation à plusieurs ainsi que la notifications des membres d'un salon lors de l'arrivée et du départ d'un autre membre est automatiquement gérée par le serveur. Ce qui n'est pas le cas avec les groupes.

4.4.4.2 Restriction des propriétés

Nous avons limité les modifications des propriétés des salons et des utilisateurs, car les communautés ne se gèrent pas de la même manière que des salons ordinaires.

Les communautés Les propriétés des communautés que nous imposons sont les suivantes :
- Un salon ne peut être crée, modifier et supprimer que par l'administrateur des communautés.
- Tous les salons sont visibles, nous voulions que ceux-ci ne soient visibles que par leur membres, cependant cette fonctionnalité n'est pas encore implémenté dans Wildfire. Nous avons d'ailleurs participé à un vote sur le site de wildfire pour que cette nouvelle fonctionnalité soit implémentée au plus vite.
- L'accès à un salon est réservé uniquement aux membres de ce salon.
- Un utilisateur membre d'un salon ne peut inviter un autre utilisateur dans le salon.
- Une communautée instantanée n'a aucune restriction au niveau du nombre de ses membres

Affectation des utilisateurs Les propriétés des utilisateurs que nous imposons sont les suivantes :
- Un utilisateur ne peut ni modifier son login ni son nom.
- Le nom de l'utilisateur est imposé, il est composé d'un préfixe, déterminé par l'administrateur, et de son login. Le préfixe permet de différencier un utilisateur de communautés instantanées des autres utilisateurs.

4.4.5 Internationalisation de la console

Pour faciliter l'évolution du plugin, nous l'avons développé en utilisant les possibilités d'internationalisation offerte par Java. Ainsi l'ensemble des pages de la console d'administration du plugin reste facilement portable vers d'autres langues.

4.4.6 Problèmes rencontrés

4.4.6.1 Changement fréquents de version du serveur Wildfire

JiveSoftware développe régulièrement de nouvelles versions de Wildfire, nous devions donc vérifier la compatibilité descendante de notre plugin, même si au départ il était prévu de le développer que pour la version 2.5 (version 2.6.2 actuellement disponible).

4.4.6.2 Console d'administration

Nous n'avons pas rencontré de problèmes majeurs au cours du développement de la console d'administration. Cette phase a surtout été longue et fastidieuse à cause du manque de documentation à ce sujet. Le seul problème que nous ayons eu concerne l'internationalisation de la console. En effet, nous avions tout d'abord utilisé les fichiers de traduction de wildfire. Cependant lorsque que nous avons testé notre plugin avec la version 2.6.2, la console ne fonctionnait plus car ceux-ci avaient changé. Nous avons donc du créer notre propre fichier de traduction afin que l'internationalisation fonctionne quelque soit la version de Wildfire.

4.5 Répartition des tâches

Nous avons réparti les tâches de notre projet en 4 unités de travail :
- les clients et les tests associés
- le plugin Wildfire avec sa console d'administration et sa console de tests
- les Web Services
- la documentation et la recherche d'un client XMPP pour Pocket PC (supportant les salons de discussion)

Les diagrammes de secteurs qui suivent représentent comment le groupe s'est réparti les differentes tâches énumérées ci-dessus.

Nous pouvons remarquer que Michèle Reynier et Jean-Michael Legait se sont focalisés sur les clients. (cf diagramme 4.12)

Michèle Barré et Christophe Roger se sont principalement occupé du plugin Wildfire, de sa console d'administration et de sa console de tests (cf diagramme 4.11).

Ilya Naraghi a surtout travaillé sur les web services. Toute l'équipe a du se documenter mais Ilya y a passé plus de temps car les web services étaient une nouvelle techonolgie pour nous tous. 4.13

Et chacun d'entre nous a rédigé de la documentation et cherché un client XMPP pour Pocket PC compatible avec les salons.

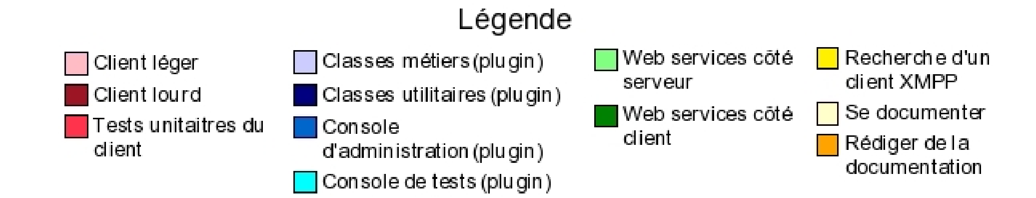


FIG. 4.10 – Légende

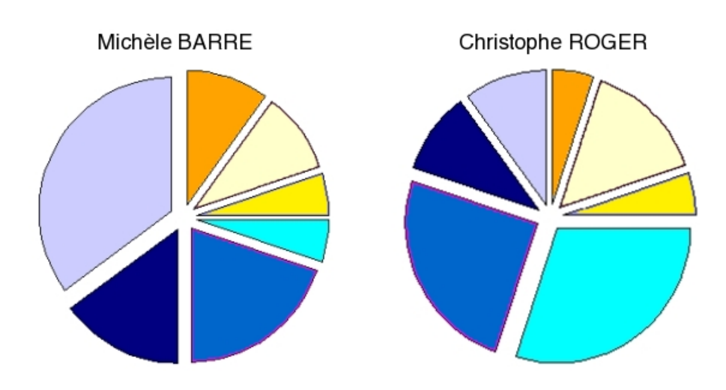


FIG. 4.11 – Michèle Barre et Christophe Roger ont surtout travaillé sur le plugin Wildfire.

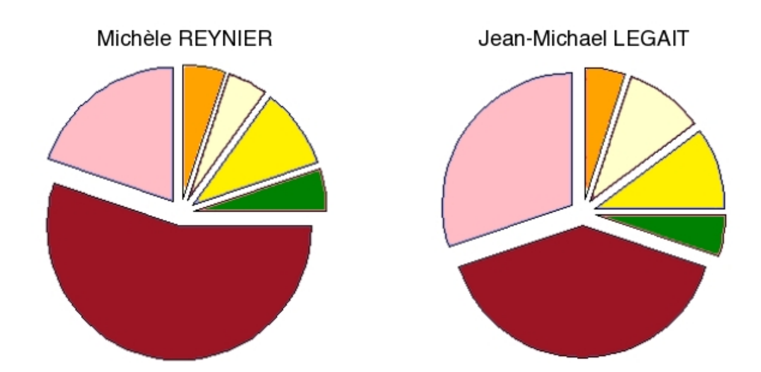


FIG. 4.12 – Michèle Reynier et Jean-Michael Legait ont surtout travaillé sur les clients.

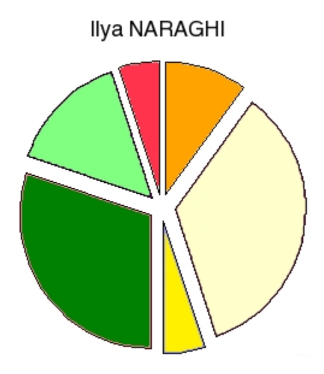


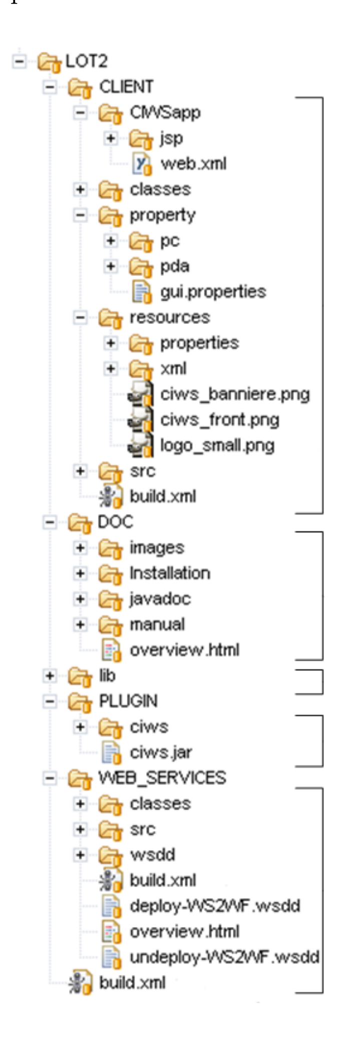
FIG. 4.13 – Ilya Naraghi a surtout travaillé sur les Web services.

Chapitre 5

Contenu du livrable

Remarque *Le livrable n'est pas fourni avec ce rapport, il se situe sur une base CVS de l'I3S, qui a été mis à notre disposition par notre encadrant Mr Collet.*
Les sources ne peuvent être fournies car elles vont être utilisées dans le cadre d'un contrat de recherche entre le laboratoire I3S et FRANCE TELECOM.

Voici l'arborescence des fichiers que nous livrons :



Le répertoire se découpe en cinq sous-répertoires qui seront décrits dans les sections suivantes.

5.1 Répertoire client

Dans ce répertoire vous trouverez un script Ant *build.xml*. Ce fichier permet de lancer différentes cibles, dont voici le récapitulatif :

Cible	Description
clean	Supprime le répertoire "classes"
veryclean	Supprime le repertoire "classes", et les clients générés.
compile	Compile toutes les sources du client
compileWSP	Compile les stubs générés par WSDL2Java
WSDL2Java	Génére les stubs grâce au fichier WSDL su serveur
clientTest	Lance les tests unitaires du client
war	Créer un répertoire "clientweb" contenant une application web. Il s'agit du client léger
jarPC	Créer un répertoire "clientpc" contenant un jar, un répertoire lib, et un répertoire save. Il s'agit du client lourd pour PC.
runJarPC	Créer puis execute le client lourd pour PC.
jarPDA	Créer un répertoire "clientpda" contenant un jar, un répertoire lib, et un répertoire save. Il s'agit du client lourd pour Pocket PC.
runJarPDA	Créer puis execute le client lourd pour PDA.
allJars	Créer un client lourd pour PC, et un autre pour Pocket PC.
all	Créer tout les clients.

Le répertoire *src* contient toutes les sources des classes du client. Voici les différents package :
- fr.unice.amui.client
- fr.unice.amui.test
- fr.unice.amui.web
- fr.unice.amui.ws

Le répertoire *CIWSapp* contient les fichiers *JSP*, ainsi qu'un fichier de description de deploiement (*web.xml*) du client léger.

Le répertoire *property* contient les fichiers de propriétés qui permettent de décrire quel web service utiliser en fonction de la plate-forme matérielle utilisée. Il possède également un fichier permettant de modifier la langue utilisé dans l'interface graphique.

Le répertoire *resources* contient les fichiers dans lesquels seront sauvés les mots clés de l'utilisateur. Il contient également les différentes images de l'interface graphique.

5.2 Répertoire Web_Services

Cible	Description
clean	Supprime le répertoire "classes"
veryclean	Supprime le repertoire "classes".
compile	Compile toutes les sources du Web services
deploy-WS2WF	Deploie le Web Service Ws2Wf au près d'Axis.
undeploy-WS2WF	Supprime le Web Service Ws2Wf au près d'Axis.

5.3 Le répertoire plugin

Le répertoire *ciws* contient l'arborescence nécessaire à l'implémentation d'un plugin. Ce répertoire peut être déposé tel quel dans src/plugin des sources de WildFire. Le plugin pourra alors être compilé en faisant la commande `ant plugins`, en utilisant le *build.xml* fourni avec les sources de wildfire.

ciws.jar est le plugin compilé et utilisable qu'il suffit de déployer sur le serveur Wildfire.

5.4 Le répertoire doc

Ce répertoire contient la Javadoc du projet, ainsi que le manuel de conception et le manuel d'installation.

5.5 Le répertoire lib

Ce répertoire contient toutes les bibliotèques nécessaires au projet.

Chapitre 6

Synthèse

6.1 Bilan technique

6.1.1 Bilan fonctionnel

Le tableau suivant récapitule les fonctionnalités que nous avions prévu d'implémenter lors de la phase de gestion de projet et indique si elles ont été réalisés ou non. Il montre aussi celles qui n'étaient pas prévues à l'origine et qui ont été rajoutées par la suite.

priorité	objectif	réalisé
1	Connexion à l'aide d'un login et d'un mot de passe	✓
1	Affectation aux salons (String.equals)	✓
1	Notification lors de l'arrivée et du départ d'un autre membre [1]	~
1	Console d'administration correspondante sur le plugin Wildfire	✓
2	Vérification des logins redondants	✓
2	Gestion des déconnexions [2]	~
2	Console d'administration correspondante sur le plugin Wildfire	✓
2	Console de tests du plugin Wildfire	✓
3	Client lourd	✓
4	Fonctionnalités de sauvegarde associées au client lourd	✓
5	Analyse par proximité des mots-clefs	✓
	Tests unitaires du client	+
	Differenciation entre les communautés et les salons ordinaires dans la console d'administration	+
	Propriétés modifiables du plugin dans un fichier de propriétés	+
	Internationalisation du client et du plugin	+
	Recherche des mots-clefs dans un index qui contient toute les informations relatives à un salon	+

légende :
✓ fait
~ devenue inutile
+ fait en plus de ce qui était prévu initialement

[1]Devenue inutile lorsque nous avons choisi d'utiliser des salons de discussion à la place des groupes de discussions.
[2]Nous avons, par la suite, décidé de changer de politique en accord avec nos encadrants et de laisser la suppression des comptes sous la responsabilité de l'administrateur.

6.1.2 Bilan non fonctionnel

6.1.2.1 Documentation

Manuel de conception Nous avons rédigé un manuel de conception (en html) que nous avons intégré à la javadoc du projet. Il explique l'architecture générale du projet (Vous trouverez ce manuel en annexe du rapport).

Manuel d'installation L'installation du serveur étant longue et compliquée, nous avons rédigé un manuel explicatif.

6.1.3 Vérification et Validation

Pour nous assurer du bon comportement de nos programmes, nous avons développé des tests unitaires pour chacun des éléments de la chaîne de fonctionnement.

Afin de valider notre projet, nous avons régulièrement rencontré nos encadrants pour leur montrer l'avancement de notre travail et leur fournir des lots fonctionnels.

6.2 Bilan personnel

Pour commencer, toute l'équipe a trouvé ce TER très intéressant sur de nombreux points.

Tout d'abord il s'appuie sur des nouvelles technologies que nous ne connaissions pas. Nous avons donc été obligé de nous auto-former en lisant de la documentation, ou encore en postant sur des forums officiels.

La complexité du projet et la taille de notre équipe de développement nous ont imposé de passer beaucoup de temps sur la gestion de projet. Cette tâche qui nous a longtemps paru ennuyeuse, a rapidement montré son importance puisqu'elle nous a permis de nous fixer des objectifs à remplir pour chacune des étapes du projet et ainsi d'éviter d'accumuler du retard.

L'importance de notre équipe, nous a également poussé à beaucoup communiquer et à utiliser des outils de travail collaboratif tels que la base CVS mise à disposition par nos encadrants. Elle s'est d'ailleurs rapidement révélée indispensable.

Enfin, il est gratifiant de savoir que ce TER est la base d'un projet plus vaste avec des objectifs industriels (pour France Telecom). Il sera d'ailleurs suivi d'un stage auquel participera Ilya Naraghi.

Bibliographie

[1] Tomcat, http ://tomcat.apache.org/

[2] Apache Axis, http ://ws.apache.org/axis/

[3] Mysaifu, http ://www2s.biglobe.ne.jp/ dat/java/project/jvm/index_en.html

[4] KSOAP, http ://ksoap.objectweb.org/software/downloads/index.html

[5] GnuClasspath, http ://www.gnu.org/software/classpath/

[6] GnuClasspathX, http ://www.gnu.org/software/classpathx/

[7] J2SE, http ://java.sun.com/javase/index.jsp

[8] J2ME, http ://java.sun.com/javame/index.jsp

[9] JUnit, http ://www.junit.org/index.htm

[10] Trucs et astuces pour pocketPC, Michel Buffa, http ://miageprojet.unice.fr/twiki/bin/view/Fun/PocketPC

[11] Jabber, http ://jabber.org

[12] Wildfire, http ://www.jivesoftware.org/

[13] API JiveSoftware Wildfire, http ://www.jivesoftware.org/builds/wildfire/docs/latest/documentation/javadoc/

[14] Apache Lucene, http ://lucene.apache.org/java/docs/index.html

[15] API Apache Lucene, http ://lucene.apache.org/java/docs/api/index.html

Annexe A

Définitions et acronymes

A.1 Web Services

Mécanisme "universel" pour faire dialoguer deux applications au travers du réseau Internet, les **services Web** reposent sur des standards. Les services Web proposent un mécanisme de communication standard pour faire dialoguer deux applications basées sur des technologies hétérogènes. La communication repose, le plus souvent, sur l'échange de messages XML. L'architecture le plus couramment utilisée est orientée services (SOA), reposant sur un mécanisme de type RPC. Les messages sont véhiculés via **SOAP** ou **XML-RPC**. La plupart des standards et technologies les plus anciennes ont été conçus pour une architecture reposant sur **SOAP**. Extrêmement nombreux, les standards des **services Web** sont organisés en couches qui n'évoluent pas à la même vitesse. Seules les fondations - **SOAP** et **WSDL** - sont aujourd'hui matures et stables.

A.2 Protocole SOAP

SOAP (Simple Object Access Protocol) définit un protocole permettant des appels de procédures à distances (**RPC**) s'appuyant principalement sur le protocole HTTP et sur XML, mais aussi SMTP et POP. Il permet ainsi de définir des **services Web** . Les paquets de données circulent sous forme de texte structuré au format XML

A.3 WSDL

Le **WSDL** est le langage de description de web services, permettant aux applications les utilisant d'auto configurer les échanges entre eux.

Le **WSDL** (Web Services Description Language) est, comme son nom l'indique, un langage de description de **Web Services** , au format XML. Il permet de séparer la description des fonctionnalités abstraites offertes par un service, des détails concrets d'une description de service, tels que "comment" et "où" cette fonctionnalité est proposée. C'est donc un langage décrivant les fonctionnalités abstraites d'un service ainsi que l'architecture décrivant les détails concrets de la description de service. En clair, il définit, de manière abstraite et indépendante du langage, l'ensemble des opérations et des messages qui peuvent être transmis vers et depuis un service web donné. Le WSDL décrit quatre ensembles de données importants : - information d'interface décrivant toutes les fonctions disponibles publiquement, - information de type de donnée pour toutes les requêtes de message et requêtes de réponse, - information de liaison sur le protocole de transport utilisé, - information d'adresse pour localiser le service spécifié.

WSDL est donc conçu pour être la pierre d'angle de l'édifice Web Services, avec un langage commun pour décrire les services et une plateforme pour intégrer automatiquement ces services.

A.4 AXIS

AXIS est l'acronyme de Apache eXtensible Interaction System . **Apache Axis** est une nouvelle implémentation de la spécification **SOAP** (Simple Object Access Protocol) développé par la fondation Apache (The Apache Software Foundation), qui succède à Apache SOAP. **Axis** se veut elle plus performante, plus modulaire et plus extensible que son prédécesseur. Axis est à la fois un environnement d'hébergement de services Web, et un "toolkit" complet de développement pour la création de services et l'accès à des services tiers.

A.5 WSDD

Fichier XML pour déployer un service Web manuellement et sans posséder forcement les sources.

A.6 Wildfire

JiveSoftware Wildfire est un serveur de messagerie instantanée d'entreprise développé sous licence Open Source GPL et sous licence commerciale. Il utilise le seul protocole open source très largement répandu pour faire de la messagerie instantanée, XMPP (également appelé Jabber). Wildfire est facile à installer et à administrer, mais offre des garanties de sécurité et de performance.

A.7 Spark

Spark est un client de messagerie instantanée respectant le protocole XMPP.

A.8 Protocole XMPP

XMPP est un sigle signifiant " eXtensible Messaging and Presence Protocol " (Protocole extensible de présence et de messagerie). C'est le nom d'un standard de l'**IETF** en développement constant, ouvert et basé sur XML. L'implémentation la plus connue est **Jabber**.

A.9 Jabber

Jabber est un système standard et ouvert de messagerie instantanée sécurisée et sans spam, de notification de présence, de collaboration et d'échange multimédia. **Jabber** est un ensemble de protocoles fondé sur le langage XML. Des logiciels fondés sur **Jabber** sont déployés sur des milliers de serveurs sur l'Internet et sont utilisés par plus de dix millions d'utilisateurs ([1] en septembre 2003, soit bien avant l'ouverture de Google Talk) à travers le monde. Le protocole lui-même est maintenu par la **Jabber Software Foundation** et est standardisé par l'**IETF** sous le nom **XMPP**. À la différence des autres systèmes de présence et de messagerie instantanées populaires, **Jabber** est conçu de manière plus large et ouverte que le simple " chat ". **Jabber** est ainsi également utilisé par les entreprises et administrations dans le cadre d'échange de données entre applications (ETL, EAI, ESB) au sein des systèmes d'informations, mais aussi dans le cadre du grid computing, des notifications d'alertes ou d'informations, de la supervision et du monitoring système et réseau. Jeremie Miller a commencé le projet en 1998 et la première version publique est sortie en mai 2000. La principale production du projet est **jabberd** , un serveur permettant aux logiciels clients de se connecter pour discuter. Ce serveur permet soit de créer un réseau Jabber privé (derrière un pare-feu), soit de rejoindre d'autres serveurs publics sur Internet, pour dialoguer en ligne avec ses correspondants.

A.10 IETF

L'**Internet Engineering Task Force** , abrégée IETF, littéralement traduit de l'anglais en " Détachement d'ingénierie d'Internet " est un groupe informel, international, ouvert à tout individu, qui participe à l'élaboration de standards pour Internet. L'**IETF** produit la plupart des nouveaux standards d'Internet.

A.11 Client XMPP

Un **client de messagerie XMPP** est un logiciel qui permet de se connecter à un serveur de messagerie utilisant le protocole **XMPP**.

A.12 PDA

Personal Digital Assistant ou Assistant Personnel Numérique.

A.13 Imov

imov Messenger est un client de messagerie instantanée **Jabber (XMPP)** fonctionnant sous Windows Mobile. **imov Messenger**, basé sur des standards ouverts, est aussi bien compatible avec des systèmes de messagerie instantanée propriétaires qu'avec la plateforme serveur ouverte **Jabber**.

A.14 Mysaifu

Mysaifu est machine virtuelle java (JVM) prévue pour tourner sous Windows Mobile 2003 pour Pocket pc. C'est un logiciel gratuit publié sous la licence **GPLv2** (GNU Public License version 2).

A.15 Apache Tomcat

Le serveur **Apache Tomcat** est un serveur Open Source qui agit comme un conteneur de servlet **J2EE**. Il fait partie du projet **Jakarta** , au sein de la fondation Apache. **Tomcat** implémente les spécifications des servlets et des JSP de **Sun Microsystems**. Comme **Tomcat** inclut un serveur HTTP interne, il est aussi considéré comme un serveur HTTP.

Annexe B

Clients XMPP testés

Voici le récapitulatif des clients XMPP testé et le problème correspondant :

B.1 Clients XMPP fonctionnant sur WinCE

Client	Problème
imov	Ne supporte pas les salons de discussions
EntreatCE	Permet d'envoyer des messages, mais pas d'en recevoir

B.2 Clients Java XMPP

Nous avons testé Jeti qui provoquait une erreur indiquant que le Pocket PC ne dispoqait pas d'assez de mémoire.

B.3 Clients XMPP sur navigateurs internet

Nous avons testé JWChat avec les navigateurs suivants :

Navigateur	Problème
Internet Explorer Webby	Ne permet pas de cliquer sur le bouton login
Opera Minimo	Permet d'envoyer des messages, mais pas d'en recevoir

B.4 Clients XMPP fonctionnant sur une J2ME

Client	Problème
Bombus Colibri JabberMixClient KomKom mober Papla Mobile	Nous n'avons pas eu le temps d'installer une J2ME sur le Pocket PC